intel

Extensible Firmware Interface Specification

Errata for Review Draft Version 0.95

Version 1.10

October 4, 2002

Intel order number: xxxxxx-001

History

Draft Revision	Draft Revision History	Date
0.95 Errata	**General:** **1.** Initial draft of errata	10/4/02

Table of Contents

Introduction

This document contains the additions, changes, clarifications, and corrections to draft 0.95 of the *Extensible Firmware Interface Specification,* version 1.10 (hereafter referred to as the "EFI 1.10 Specification"). This document is intended to be used along with the EFI 1.10 Specification, draft 0.95, that is available from the EFI Web Site at:
http://developer.intel.com/technology/efi.

Additions and Changes

Section 2.3.2: IA-32 Platforms (Page 2-8)

For the bulleted list that starts with, "For an operating system to use any EFI runtime services, it must:," add the following bullets at the end of the list:

- ACPI Tables must be contained in memory of type **EfiACPIReclaimMemory**.
- The system firmware must not request a virtual mapping for any memory descriptor of type **EfiACPIReclaimMemory** or **EfiACPIMemoryNVS**.
- EFI memory descriptors of type **EfiACPIReclaimMemory** and **EfiACPIMemoryNVS** must be aligned on a 4 KB boundary and must be a multiple of 4 KB in size.
- Any EFI memory descriptor that requests a virtual mapping via the **EFI_MEMORY_DESCRIPTOR** having the **EFI_MEMORY_RUNTIME** bit set must be aligned on a 4 KB boundary and must be a multiple of 4 KB in size.
- An ACPI Memory Op-region must inherit cacheability attributes from the EFI memory map. If the system memory map does not contain cacheability attributes, the ACPI Memory Op-region must inherit its cacheability attributes from the ACPI name space. If no cacheability attributes exist in the system memory map or the ACPI name space, then the region must be assumed to be noncacheable.
- An ACPI Memory Op-region must inherit cacheability attributes from the EFI memory map. If the EFI memory map does not contain cacheability attributes, the ACPI Memory Op-region must inherit its cacheability attributes from the ACPI name space. If no cacheability attributes exist in the system memory map or the ACPI name space, then the region must be assumed to be noncacheable.

Section 2.3.3: Itanium™-based Platforms (Page 2-9)

After the sentence, "If **SetVirtualAddressMap()** has been used, then runtime service calls are made in virtual mode," add the following bulleted list:

- ACPI Tables must be contained in memory of type **EfiACPIReclaimMemory**.
- The system firmware must not request a virtual mapping for any memory descriptor of type **EfiACPIReclaimMemory** or **EfiACPIMemoryNVS**.
- EFI memory descriptors of type **EfiACPIReclaimMemory** and **EfiACPIMemoryNVS** must be aligned on an 8 KB boundary and must be a multiple of 8 KB in size.
- Any EFI memory descriptor that requests a virtual mapping via the **EFI_MEMORY_DESCRIPTOR** having the **EFI_MEMORY_RUNTIME** bit set must be aligned on a 8 KB boundary and must be a multiple of 8 KB in size.
- An ACPI Memory Op-region must inherit cacheability attributes from the EFI memory map. If the system memory map does not contain cacheability attributes the ACPI Memory Op-region must inherit its cacheability attributes from the ACPI name space. If no cacheability attributes exist in the system memory map or the ACPI name space, then the region must be assumed to be noncacheable.
- An ACPI Memory Op-region must inherit cacheability attributes from the EFI memory map. If the EFI memory map does not contain cacheability attributes, the ACPI Memory Op-region must inherit its cacheability attributes from the ACPI name space. If no cacheability attributes exist in the system memory map or the ACPI name space, then the region must be assumed to be noncacheable.

Clarifications and Corrections

Section 2.5.6: Platform Components (Page 2-24)

Add the following paragraph between the first and second paragraph of this section:

> Since the platform firmware may choose to only connect the devices required to produce consoles and gain access to a boot device, the OS present device drivers cannot assume that an EFI driver for a device has been executed. The presence of an EFI driver in the system firmware or in an option ROM does not guarantee that the EFI driver will be loaded, executed, or allowed to manage any devices in a platform. All OS present device drivers must be able to handle devices that have been managed by an EFI driver and devices that have not been managed by an EFI driver.

Section 5: Services – Boot Services

Additions and Changes

Section 5.2: Memory Allocation Services (Page 5-18)

Add the following sentence to the end of the last paragraph on page 5-18:

> The system firmware must follow the processor-specific rules outlined in sections 2.3.2 and 2.3.3 in the layout of the EFI memory map to enable the OS to make the required virtual mappings.

Section 5.2: AllocatePages() – Parameters – MemoryType (Page 5-21)

Change the description of the *MemoryType* parameter FROM:

> The type of memory to allocate. The only types allowed are **EfiLoaderCode, EfiLoaderData, EfiRuntimeServicesCode, EfiRuntimeServicesData, EfiBootServicesCode, EfiBootServicesData, EfiACPIReclaimMemory, EfiACPIMemoryNVS**, and **EfiReservedMemoryType**. Normal allocations (that is, allocations by any EFI application) are of type **EfiLoaderData**. See "Related Definitions", Table5-5, and Table 5-6.

TO:

> The type of memory to allocate. The type **EFI_MEMORY_TYPE** is defined in "Related Definitions" below. These memory types are also described in more detail in Table 5-5 and Table 5-6. Normal allocations (that is, allocations by any EFI application) are of type **EfiLoaderData**. *MemoryType* values in the range 0x80000000..0xFFFFFFFF are reserved for use by EFI OS loaders that are provided by operating system vendors. The only illegal memory type values are those in the range **EfiMaxMemoryType**..0x7FFFFFFF.

Section 5.2: AllocatePages() – Status Codes Returned (Page 5-23)

Change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The requested pages were allocated.
EFI_OUT_OF_RESOURCES	The pages could not be allocated.
EFI_INVALID_PARAMETER	One of the parameters has an invalid value.
EFI_NOT_FOUND	The requested pages could not be found.

TO:

EFI_SUCCESS	The requested pages were allocated.
EFI_OUT_OF_RESOURCES	The pages could not be allocated.
EFI_INVALID_PARAMETER	*Type* is not **AllocateAnyPages** or **AllocateMaxAddress** or **AllocateAddress**.
EFI_INVALID_PARAMETER	*MemoryType* is in the range **EfiMaxMemoryType**..0x7FFFFFFF.
EFI_NOT_FOUND	The requested pages could not be found.

Section 5.2: AllocatePool() – Parameters – PoolType (Page 5-29)

Change the description of the parameter *PoolType* FROM:

The type of pool to allocate. The only supported types are **EfiLoaderData, EfiBootServicesData, EfiRuntimeServicesData, EfiACPIReclaimMemory,** and **EfiACPIMemoryNVS**. Type **EFI_MEMORY_TYPE** is defined in the **AllocatePages()** function description.

TO:

The type of pool to allocate. Type **EFI_MEMORY_TYPE** is defined in the **AllocatePages()** function description. *PoolType* values in the range 0x80000000..0xFFFFFFFF are reserved for use by EFI OS loaders that are provided by operating system vendors. The only illegal memory type values are those in the range **EfiMaxMemoryType**..0x7FFFFFFF.

Section 5.4: StartImage() (Page 5-78)

Add the following section, "EFI 1.10 Extension," between the "Description" and "Status Codes Returned" sections:

EFI 1.10 Extension

To maintain compatibility with EFI drivers that are written to the *EFI 1.02 Specification*, **StartImage()** must monitor the handle database before and after each image is started. If any handles are created or modified when an image is started, then **ConnectController()** must be called for each of the newly created or modified handles before **StartImage()** returns.

Clarifications and Corrections

Section 5.1: CheckEvent() – Status Codes Returned (Page 5-12)

Change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The event is in the signaled state.
EFI_NOT_READY	The event is not in the signaled state.

TO:

EFI_SUCCESS	The event is in the signaled state.
EFI_NOT_READY	The event is not in the signaled state.
EFI_INVALID_PARAMETER	*Event* is of type **EVT_NOTIFY_SIGNAL**.

Section 5.2: Memory Allocation Tables – Table 5-5 – EfiMemoryMappedIoPortSpace (Page 5-19)

Change the description of "EfiMemoryMappedIoPortSpace" in Table 5-5 FROM:

System memory mapped IO region that is used to translate memory cycles to IO cycles.

TO:

System memory mapped IO region that is used to translate memory cycles to IO cycles by the processor.

Section 5.2: Memory Allocation Tables – Table 5-5 (Page 5-19)

Add the following note below Table 5-5:

NOTE

*There is only one region of type **EfiMemoryMappedIoPortSpace** defined in the architecture for Itanium® processor family–based platforms. As a result, there should be one and only one region of type **EfiMemoryMappedIoPortSpace** in the EFI memory map of an Itanium processor family–based platform.*

Section 5.2: GetMemoryMap() – Related Definitions (Page 5-26)

Change the description of the *NumberOfPages* field of the **EFI_MEMORY_DESCRIPTOR** FROM:

Number of pages in the memory region.

TO:

Number of 4 KB pages in the memory region.

Section 5.3.1: InstallProtocolInterface() – Description (Page 5-37)

Add the following sentence to the end of the first paragraph in the "Description" section:

> The same GUID cannot be installed more than once onto the same handle. If the same GUID is installed more than once onto the same handle, then the results are not predictable.

Section 5.3.1: RegisterProtocolNotify() – Parameters – Event (Page 5-42)

Change the description of the *Event* parameter FROM:

> Event that is to be signaled whenever a protocol interface is registered for *Protocol*. Type **EFI_EVENT** is defined in the **InstallProtocolInterface()** function description.

TO:

> Event that is to be signaled whenever a protocol interface is registered for *Protocol*. The type **EFI_EVENT** is defined in the **InstallProtocolInterface()** function description. The same **EFI_EVENT** may be used for multiple protocol notify registrations.

Section 5.3.1: OpenProtocolInformation() – Description (Page 5-58)

Change the first sentence of the third paragraph FROM:

> If the interface specified by *Protocol* is supported by the handle specified by *Handle*, then *EntryBuffer* is allocated with the boot service **AllocatePages()**, and *EntryCount* is set to the number of entries in *EntryBuffer*.

TO:

> If the interface specified by *Protocol* is supported by the handle specified by *Handle*, then *EntryBuffer* is allocated with the boot service **AllocatePool()**, and *EntryCount* is set to the number of entries in *EntryBuffer*.

Section 5.3.1: DisconnectController() – Description (Page 5-64)

Change the sixth sentence of the first paragraph FROM:

> If *ChildHandle* is the only child of *ControllerHandle*, then the drivers specified by *DriverImageHandle* will be disconnected from *ControllerHandle*.

TO:

> If *ChildHandle* is the only child of *ControllerHandle*, then the driver specified by *DriverImageHandle* will be disconnected from *ControllerHandle*.

Section 5.3.1: LocateHandleBuffer() – Prototype (Page 5-68)

Change the "Prototype" section FROM:

```
typedef
EFI_STATUS
LocateHandle (
  IN EFI_LOCATE_SEARCH_TYPE  SearchType,
  IN EFI_GUID                *Protocol   OPTIONAL,
  IN VOID                    *SearchKey  OPTIONAL,
  IN OUT UINTN               *NoHandles,
  OUT EFI_HANDLE             **Buffer
  );
```

TO:

```
typedef
EFI_STATUS
LocateHandleBuffer (
  IN EFI_LOCATE_SEARCH_TYPE  SearchType,
  IN EFI_GUID                *Protocol   OPTIONAL,
  IN VOID                    *SearchKey  OPTIONAL,
  IN OUT UINTN               *NoHandles,
  OUT EFI_HANDLE             **Buffer
  );
```


Section 5.3.1: InstallMultipleProtocolInterfaces() – Description (Page 5-72)

Add the following to the end of the first paragraph in the "Description" section:

> The same GUID cannot be installed more than once onto the same handle. If the same GUID is installed more than once onto the same handle, then the results are not predictable.

Section 5.4: StartImage() – Parameters – ExitDataSize (Page 5-78)

Change the description of the *ExitDataSize* parameter FROM:

> Pointer to the size, in bytes, of *ExitData*.

TO:

> Pointer to the size, in bytes, of *ExitData*. If *ExitData* is **NULL**, then this parameter is ignored and the contents of *ExitDataSize* are not modified.

Section 5.5: CopyMem() – Description (Page 5-87)

Add the following paragraph to the end of the "Description" section:

The contents of the *Destination* buffer on exit from this service must match the contents of the *Source* buffer on entry to this service. Due to potential overlaps, the contents of the *Source* buffer may be modified by this service. The following rules can be used to guarantee the correct behavior:

1. If *Destination* and *Source* are identical, then no operation should be performed.
2. If *Destination* > *Source* AND *Destination* < (*Source* + *Length*), then the data should be copied from the *Source* buffer to the *Destination* buffer starting from the end of the buffers and working toward the beginning of the buffers.
3. Otherwise, the data should be copied from the *Source* buffer to the *Destination* buffer starting from the beginning of the buffers and working toward the end of the buffers.

Section 5.5: CalculateCrc32() – Parameters – Crc32 (Page 5-92)

Change the description of the parameter *Crc32* FROM:

The 32 bit CRC that was computer for the data buffer specified by *Data* and *DataSize*.

TO:

The 32-bit CRC that was computed for the data buffer specified by *Data* and *DataSize*.

Section 6: Services – Runtime Services

Additions and Changes

Section 6.3: Virtual Memory Services (Page 6-16)

Add the following sentence to the end of the first paragraph on page 6-16:

> The system firmware must follow the processor-specific rules outlined in sections 2.3.2 and 2.3.3 in the layout of the EFI memory map to enable the OS to make the required virtual mappings.

Clarifications and Corrections

Section 6.2: GetWakeupTime() – Status Codes Returned (Page 6-14)

Change the "Status Codes Returned" table FROM:

EFI_SUCCESS	The alarm settings were returned.
EFI_INVALID_PARAMETER	A time field is out of range.
EFI_DEVICE_ERROR	The wakeup time could not be retrieved due to a hardware error.
EFI_UNSUPPORTED	A wakeup timer is not supported on this platform.

TO:

EFI_SUCCESS	The alarm settings were returned.
EFI_INVALID_PARAMETER	*Enabled* is **NULL**.
EFI_INVALID_PARAMETER	*Pending* is **NULL**.
EFI_INVALID_PARAMETER	*Time* is **NULL**.
EFI_DEVICE_ERROR	The wakeup time could not be retrieved due to a hardware error.
EFI_UNSUPPORTED	A wakeup timer is not supported on this platform.

Section 6.3: SetVirtualAddressMap() – Description (Page 6-18)

Change the contents of the fourth paragraph of the "Description" section FROM:

> In addition, all of the fields of the EFI Runtime Services Table and several fields of the EFI System Table must be converted from physical pointers to virtual pointers using the **ConvertPointer()** function. The EFI System Table fields include *FirmwareVendor*, *RuntimeServices*, and *ConfigurationTable*. Because fields of both the EFI Runtime Services Table and the EFI System Table are being

modified, the 32-bit CRC for the EFI Runtime Services Table and the EFI System Table must be recomputed.

TO:

In addition, all of the fields of the EFI Runtime Services Table except *SetVirtualAddressMap* and *ConvertPointer* must be converted from physical pointers to virtual pointers using the **ConvertPointer()** service. The **SetVirtualAddressMap()** and **ConvertPointer()** services are only callable in physical mode, so they do not need to be converted from physical pointers to virtual pointers. Several fields of the EFI System Table must be converted from physical pointers to virtual pointers using the **ConvertPointer()** service. These fields include *FirmwareVendor*, *RuntimeServices*, and *ConfigurationTable*. Because contents of both the EFI Runtime Services Table and the EFI System Table are modified by this service, the 32-bit CRC for the EFI Runtime Services Table and the EFI System Table must be recomputed.

Section 6.4: ResetSystem() – Description (Page 6-22)

Change the third and fourth paragraphs FROM:

Calling this interface with *ResetType* of **EfiResetWarm** causes a system-wide initialization. The processors are set to their initial state, and pending cycles are not corrupted. If the system does not support this reset type, then an **EfiColdReset** must be performed.

Calling this interface with *ResetType* of **EfiResetShutdown** causes power to be removed from the system. If the system does not support this reset type, then an **EfiColdReset** must be performed. If an ACPI S5 method is available to remove power from the system, then this reset type should not be used.

TO:

Calling this interface with *ResetType* of **EfiResetWarm** causes a system-wide initialization. The processors are set to their initial state, and pending cycles are not corrupted. If the system does not support this reset type, then an **EfiResetCold** must be performed.

Calling this interface with *ResetType* of **EfiResetShutdown** causes the system to enter a power state equivalent to the ACPI G2/S5 or G3 states. If the system does not support this reset type, then when the system is rebooted, it should exhibit the **EfiResetCold** attributes. If the ACPI S5 state is supported on the system, then this reset type should not be used.

Section 8: Protocols – Device Path Protocol

Additions and Changes

Section 8.3.2.2: PCCARD Device Path – Table 8-4 (Page 8-5)

Change Table 8-4 FROM:

Table 8-4. PCCARD Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 1 – Hardware Device Path
Sub-Type	1	1	Sub-Type 2 – PCCARD
Length	2	2	Length of this structure in bytes. Length is 5 bytes.
Socket Number	4	1	Socket Number (0 = First Socket)

TO:

Table 8-4. PCCARD Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 1 – Hardware Device Path
Sub-Type	1	1	Sub-Type 2 – PCCARD
Length	2	2	Length of this structure in bytes. Length is 5 bytes.
Function Number	4	1	Function Number (0 = First Function)

Section 8.3.3: ACPI Device Path (Page 8-7)

Insert the following paragraphs just before Table 8-8:

> The ACPI Device Path node only supports numeric 32-bit values for the _HID and _UID values. The Expanded ACPI Device Path node supports both numeric and string values for the _HID, _UID, and _CID values. As a result, the ACPI Device Path node is smaller and should be used if possible to reduce the size of device paths that may potentially be stored in nonvolatile storage. If a string value is required for the _HID field, or a string value is required for the _UID field, or a _CID field is required, then the Expanded ACPI Device Path node must be used. If a string field of the Expanded ACPI Device Path node is present, then the corresponding numeric field is ignored.
>
> The _HID and _CID fields in the ACPI Device Path node and Expanded ACPI Device Path node are stored as a 32-bit compressed EISA-type IDs. The following macro can be

used to compute these EISA-type IDs from a Plug and Play Hardware ID. The Plug and Play Hardware IDs used to compute the _HID and _CID fields in the EFI device path nodes must match the Plug and Play Hardware IDs used to build the matching entries in the ACPI tables.

```
#define EFI_PNP_ID(ID)  (UINT32)(((ID) << 16) | 0x41D0)
#define EISA_PNP_IP(ID) EFI_PNP_ID(ID)
```

Section 8.3.3: ACPI Device Path – Table 8-8 (Page 8-7)

Change Table 8-8 FROM:

Table 8-8. ACPI Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 2 – ACPI Device Path.
Sub-Type	1	1	Sub-Type 1 ACPI Device Path.
Length	2	2	Length of this structure in bytes. Length is 12 bytes.
_HID	4	4	Device's PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID or a String. This value must match the corresponding _HID in the ACPI name space.
_UID	8	4	Unique ID that is required by ACPI if two devices have the same _HID. This value must also match the corresponding _UID/_HID pair in the ACPI name space. Only the 32-bit numeric value type of _UID is supported; thus strings must not be used for the _UID in the ACPI name space.

TO:

Table 8-8. ACPI Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 2 – ACPI Device Path.
Sub-Type	1	1	Sub-Type 1 ACPI Device Path.
Length	2	2	Length of this structure in bytes. Length is 12 bytes.
_HID	4	4	Device's PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID. This value must match the corresponding _HID in the ACPI name space.
_UID	8	4	Unique ID that is required by ACPI if two devices have the same _HID. This value must also match the corresponding _UID/_HID pair in the ACPI name space. Only the 32-bit numeric value type of _UID is supported; thus strings must not be used for the _UID in the ACPI name space.

Section 8.3.3: ACPI Device Path – Table 8-9 (Page 8-8)

Change Table 8-9 FROM:

Table 8-9. Expanded ACPI Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 2 – ACPI Device Path.
Sub-Type	1	1	Sub-Type 2 Expanded ACPI Device Path.
Length	2	2	Length of this structure in bytes. Length is 16 bytes.
_HID	4	4	Device's PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID or a String. This value must match the corresponding _HID in the ACPI name space.
_UID	8	4	Unique ID that is required by ACPI if two devices have the same _HID. This value must also match the corresponding _UID/_HID pair in the ACPI name space. Only the 32-bit numeric value type of _UID is supported; thus strings must not be used for the _UID in the ACPI name space.
_CID	12	4	Device's compatible PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID or a String. This value must match at least one of the compatible device IDs returned by the corresponding _CID in the ACPI name space.

TO:

Table 8-9. Expanded ACPI Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 2 – ACPI Device Path.
Sub-Type	1	1	Sub-Type 2 Expanded ACPI Device Path.
Length	2	2	Length of this structure in bytes. Minimum length is 19 bytes. The actual size will depend on the size of the _HIDSTR, _UIDSTR, and _CIDSTR fields.
_HID	4	4	Device's PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID. This value must match the corresponding _HID in the ACPI name space.
_UID	8	4	Unique ID that is required by ACPI if two devices have the same _HID. This value must also match the corresponding _UID/_HID pair in the ACPI name space.
_CID	12	4	Device's compatible PnP hardware ID stored in a numeric 32-bit compressed EISA-type ID. This value must match at least one of the compatible device IDs returned by the corresponding _CID in the ACPI name space.

_HIDSTR	16	>=1	Device's PnP hardware ID stored as a null-terminated ASCII string. This value must match the corresponding _HID in the ACPI name space. If the length of this string not including the null-terminator is 0, then the _HID field is used. If the length of this null-terminated string is greater than 0, then this field supercedes the _HID field.
_UIDSTR	Varies	>=1	Unique ID that is required by ACPI if two devices have the same _HID. This value must also match the corresponding _UID/_HID pair in the ACPI name space. This value is stored as a null-terminated ASCII string. If the length of this string not including the null-terminator is 0, then the _UID field is used. If the length of this null-terminated string is greater than 0, then this field supercedes the _UID field. The Byte Offset of this field can be computed by adding 16 to the size of the _HIDSTR field.
_CIDSTR	Varies	>=1	Device's compatible PnP hardware ID stored as a null-terminated ASCII string. This value must match at least one of the compatible device IDs returned by the corresponding _CID in the ACPI name space. If the length of this string not including the null-terminator is 0, then the _CID field is used. If the length of this null-terminated string is greater than 0, then this field supercedes the _CID field. The Byte Offset of this field can be computed by adding 16 to the sum of the sizes of the _HIDSTR and _UIDSTR fields.

Section 8.3.4.11: InfiniBand Device Path – Table 8-20 (Page 8-12)

Change Table 8-20 FROM:

Table 8-20. InfiniBand Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 3 – Messaging Device Path
Sub-Type	1	1	Sub-Type 9 – InfiniBand
Length	2	2	Length of this structure in bytes
Reserved	4	4	Reserved
Node GUID[1]	8	8	64 bit node GUID[1] of the IOU
IOC GUID[1]	16	8	64 bit GUID[1] of the IOC
Device ID	24	8	64 bit persistent ID of the device
Note 1	The usage of the term GUID is per the Infiniband Specification. This is not the same as the **EFI_GUID** type defined in the EFI Specification.		

TO:

Table 8-20. InfiniBand Device Path

Mnemonic	Byte Offset	Byte Length	Description
Type	0	1	Type 3 – Messaging Device Path
Sub-Type	1	1	Sub-Type 9 – InfiniBand
Length	2	2	Length of this structure in bytes. Length is 48 bytes.
Resource Flags	4	4	Flags to help identify/manage InfiniBand device path elements: • Bit 0 – IOC/Service (0b = IOC, 1b = Service) • Bit 1 – Extend Boot Environment • Bit 2 – Console Protocol • Bit 3 – Storage Protocol • Bit 4 – Network Protocol All other bits are reserved.
PORT GID	8	16	128-bit Global Identifier for remote fabric port
IOC GUID/Service ID	24	8	64-bit unique identifier to remote IOC or server process. Interpretation of field specified by Resource Flags (bit 0)
Target Port ID	32	8	64-bit persistent ID of remote IOC port
Device ID	40	8	64-bit persistent ID of remote device
Note	The usage of the term GUID and GID are per the InfiniBand Specification. The term GUID is not the same as the `EFI_GUID` type defined in the EFI Specification.		

Clarifications and Corrections

None.

Section 9: Protocols – EFI Driver Model

Additions and Changes

Section 9.4: EFI_DRIVER_CONFIGURATION_PROTOCOL.SetOptions() – Parameters – ControllerHandle (Page 9-34)

Change the description of *ControllerHandle* FROM:

> The handle of the controller to set options on.

TO:

> The handle of the controller to set options on. If *ControllerHandle* is a valid **EFI_HANDLE** that is being managed by this driver, then the user will be allowed to set options for the controller specified by *ControllerHandle*. If this parameter is **NULL**, then the options will be set for all the controllers that this driver is currently managing. If *ControllerHandle* is **NULL**, then setting options for a child controller is not supported, so *ChildHandle* must also be **NULL**.

Clarifications and Corrections

None.

Section 10: Protocols – Console Support

Additions and Changes

Section 10.7: PUGA_FW_SERVICE_DISPATCH.DispatchService() – Prototype (Page 10-40)

Change the "Prototype" section FROM:

```
typedef
UGA_STATUS
(UGA_FW_CALL_TYPE_API *PUGA_FW_SERVICE_DISPATCH) (
  IN  PUGA_DEVICE              pDevice,
  IN OUT PUGA_IO_REQUEST       pIoRequest
  );
```

TO:

```
typedef
UGA_STATUS
(EFIAPI *PUGA_FW_SERVICE_DISPATCH) (
  IN  PUGA_DEVICE              pDevice,
  IN OUT PUGA_IO_REQUEST       pIoRequest
  );
```


Section 10.7: PUGA_FW_SERVICE_DISPATCH.DispatchService() – Related Definitions (Page 10-41)

Insert the following statement at the beginning of the "Related Definitions" section:

```
typedef UINT32  UGA_STATUS;
```

Clarifications and Corrections

Section 10.5: EFI_UGA_DRAW_PROTOCOL.Blt() – Prototype (Page 10-32)

Change the "Prototype" section FROM:

```
typedef
EFI_STATUS
(EFIAPI *EFI_UGA_DRAW_PROTOCOL_BLT) (
  IN EFI_UGA_DRAW_PROTOCOL     *This,
  IN OUT EFI_UGA_PIXEL          *BltBuffer,  OPTIONAL
  IN UINTN                      SourceX,
  IN UINTN                      SourceY,
  IN UINTN                      DestinationX,
  IN UINTN                      DestinationY,
  IN UINTN                      Width,
  INUINTN                       Height,
  IN UINTN                      Delta        OPTIONAL
  );
```

TO:

```
typedef
EFI_STATUS
(EFIAPI *EFI_UGA_DRAW_PROTOCOL_BLT) (
  IN EFI_UGA_DRAW_PROTOCOL     *This,
  IN OUT EFI_UGA_PIXEL          *BltBuffer,  OPTIONAL
  IN  EFI_UGA_BLT_OPERATION     BltOperation,
  IN UINTN                      SourceX,
  IN UINTN                      SourceY,
  IN UINTN                      DestinationX,
  IN UINTN                      DestinationY,
  IN UINTN                      Width,
  IN UINTN                      Height,
  IN UINTN                      Delta        OPTIONAL
  );
```

Section 11: Protocols – Bootable Image Support

Additions and Changes

Section 11.3: EFI_FILE_IO_INTERFACE.OpenVolume() – Status Codes Returned (Page 11-20)

Add the following return code to the "Status Codes Returned" section:

EFI_MEDIA_CHANGED	The device has a different medium in it or the medium is no longer supported. Any existing file handles for this volume are no longer valid. To access the files on the new medium, the volume must be reopened with **`OpenVolume()`**.

Section 11.3: EFI_FILE.Open() – Description (Page 11-24)

Change the second paragraph of the "Description" section FROM:

If **`EFI_FILE_MODE_CREATE`** is set, then the file is created in the directory. If the final location of *`FileName`* does not refer to a directory then the operation fails. If the file does not exist in the directory, then a new file is created. If the file already exists in the directory, then the existing file is deleted, and the new file is created.

TO:

If **`EFI_FILE_MODE_CREATE`** is set, then the file is created in the directory. If the final location of *`FileName`* does not refer to a directory, then the operation fails. If the file does not exist in the directory, then a new file is created. If the file already exists in the directory, then the existing file is opened.

Clarifications and Corrections

None.

Section 12: Protocols – PCI Bus Support

Additions and Changes

None.

Clarifications and Corrections

Section 12.2: EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem() – Prototype (Page 12-25)

Change the "Prototype" section FROM:

```
typedef
EFI_STATUS
(EFIAPI *EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_COPY_MEM) (
  IN     EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL        *This,
  IN     EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH  Width,
  IN     UINT64                                 Destddress,
  IN     UINT64                                 Srcddress,
  IN     UINTN                                  Count
  );
```

TO:

```
typedef
EFI_STATUS
(EFIAPI *EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_COPY_MEM) (
  IN     EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL        *This,
  IN     EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_WIDTH  Width,
  IN     UINT64                                 DestAddress,
  IN     UINT64                                 SrcAddress,
  IN     UINTN                                  Count
  );
```

Section 12.2: EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL.CopyMem() – Description (Page 12-26)

Add the following to the end of the "Description" section:

> The contents of the *DestAddress* buffer on exit from this service must match the contents of the *SrcAddress* buffer on entry to this service. Due to potential overlaps, the contents of the *SrcAddress* buffer may be modified by this service. The following rules can be used to guarantee the correct behavior:
>
> 1. If *DestAddress* > *SrcAddress* AND *DestAddress* < (*SrcAddress* + *Width* size * *Count*), then the data should be copied from the *SrcAddress* buffer to the *DestAddress* buffer starting from the end of buffers and working toward the beginning of the buffers.
> 2. Otherwise, the data should be copied from the *SrcAddress* buffer to the *DestAddress* buffer starting from the beginning of the buffers and working toward the end of the buffers.

Section 12.3.2: PCI Bus Drivers (Page 12-47)

In the paragraph after Figure 12-9, change two instances of "PCI Host Bridge I/O Protocol" to "PCI Root Bridge I/O Protocol."

Section 12.3.2: PCI Bus Drivers - Figure 12-10 (Page 12-47)

In the figure, change "PCI_ROOT_BRIDGE_PROTOCOL" to "PCI_ROOT_BRIDGE_IO_PROTOCOL."

Section 12.3.2.1: Driver Binding Protocol for PCI Bus Drivers (Page 12-48)

In the first two paragraphs, change two instances of "PCI Host Bridge I/O Protocol" to "PCI Root Bridge I/O Protocol."

Section 12.4: EFI PCI I/O Protocol (Page 12-53)

Change the second bullet in the list FROM:

- A device driver model that abstracts the I/O addresses, Memory addresses, and PCI Configuration addresses from the PCI device driver. Instead, BAR (Base Address Register) relative addressing is used for I/O and Memory accesses, and device relative addressing is used for PCI Configuration accesses.

TO:

- A device driver model that abstracts the I/O addresses, Memory addresses, and PCI Configuration addresses from the PCI device driver. Instead, BAR (Base Address Register) relative addressing is used for I/O and Memory accesses, and device relative addressing is used for PCI Configuration accesses. The BAR relative addressing is specified in the PCI I/O services as a BAR index. A PCI controller may contain a combination of 32-bit and 64-bit BARs. The BAR index represents the logical BAR number in the standard PCI configuration header starting from the first BAR. The BAR index does not represent an offset into the standard PCI Configuration Header because those offsets will vary depending on the combination and order of 32-bit and 64-bit BARs.

Section 12.4: EFI PCI I/O Protocol (Pages 12-53 to 12-90)

Change eight instances of "PCI Host Bridge I/O Protocol" to "PCI Root Bridge I/O Protocol."

Section 12.4: EFI_PCI_IO_PROTOCOL.CopyMem() – Description (Page 12-74)

Add the following to the end of the "Description" section:

The contents of the *DestOffset* buffer on exit from this service must match the contents of the *SrcOffset* buffer on entry to this service. Due to potential overlaps, the contents of the *SrcOffset* buffer may be modified by this service. The following rules can be used to guarantee the correct behavior:

1. If *DestOffset* > *SrcOffset* AND *DestOffset* < (*SrcOffset* + *Width* size * *Count*), then the data should be copied from the *SrcOffset* buffer to the *DestOffset* buffer starting from the end of buffers and working toward the beginning of the buffers.
2. Otherwise, the data should be copied from the *SrcOffset* buffer to the *DestOffset* buffer starting from the beginning of the buffers and working toward the end of the buffers.

Section 12.4: EFI_PCI_IO_PROTOCOL.Attributes() – Description (Page 12-84)

Add the following paragraph to the end of the "Description" section:

> This function will also return **EFI_UNSUPPORTED** if more than one PCI controller on the same PCI root bridge has already successfully requested one of the ISA addressing attributes. For example, if one PCI VGA controller had already requested the **VGA_IO** and **VGA_MEMORY** attributes, then a second PCI VGA controller on the same root bridge cannot succeed in requesting those same attributes. This restriction applies to the ISA-, VGA-, and IDE-related attributes.

Section 13: Protocols – SCSI Bus Support

Additions and Changes

Section 13.1: EFI_SCSI_PASS_THRU Protocol – GUID (Page 13-1)

Change the "GUID" section FROM:

```
#define EFI_SCSI_PASS_THRU_PROTOCOL_GUID \
{ 0xb881c8f5,0x4c81,0x11d4,0x80,0xb1,0x00,0xc0,0x4f,0x60,0xdf,0xee }
```

TO:

```
#define EFI_SCSI_PASS_THRU_PROTOCOL_GUID \
{ 0xa59e8fcf,0xbda0,0x43bb,0x90,0xb1,0xd3,0x73,0x2e,0xca,0xa8,0x77 }
```

Section 13.1: EFI_SCSI_PASS_THRU_PROTOCOL.PassThru() – Related Definitions – TransferLength (Page 13-6)

Change the description of *TransferLength* FROM:

> On input, the size, in bytes, of *DataBuffer*. On output, the number of bytes transferred between the SCSI controller and the SCSI device. If *TransferLength* is larger than the SCSI controller can handle, then the SCSI controller will transfer its maximum amount, and will update *TransferLength* with the number of bytes actually transferred.

TO:

> On input, the size, in bytes, of *DataBuffer*. On output, the number of bytes transferred between the SCSI controller and the SCSI device. If *TransferLength* is larger than the SCSI controller can handle, no data will be transferred, *TransferLength* will be updated to contain the number of bytes that the SCSI controller is able to transfer, and **EFI_BAD_BUFFER_SIZE** will be returned.

Section 13.1: EFI_SCSI_PASS_THRU_PROTOCOL.PassThru() – Description (Page 13-8)

Change the seventh paragraph FROM:

> If the data buffer described by *DataBuffer* and *TransferLength* is too big to be transferred in a single command, then **EFI_WARN_BUFFER_TOO_SMALL** is returned. The number of bytes actually transferred is returned in *TransferLength.*

TO:

> If the data buffer described by *DataBuffer* and *TransferLength* is too big to be transferred in a single command, then no data is transferred and **EFI_BAD_BUFFER_SIZE** is returned. The number of bytes that can be transferred in a single command are returned in *TransferLength.*

Section 13.1: EFI_SCSI_PASS_THRU_PROTOCOL.PassThru() – Status Codes Returned (Page 13-9)

Change the second return code FROM:

EFI_WARN_BUFFER_TOO_SMALL	The SCSI Request Packet was executed, but the entire *DataBuffer* could not be transferred. The actual number of bytes transferred is returned in *TransferLength*. See *HostAdapterStatus*, *TargetStatus*, *SenseDataLength*, and *SenseData* in that order for additional status information.

TO:

EFI_BAD_BUFFER_SIZE	The SCSI Request Packet was not executed. The number of bytes that could be transferred is returned in *TransferLength*. See *HostAdapterStatus*, *TargetStatus*, *SenseDataLength*, and *SenseData* in that order for additional status information.

Clarifications and Corrections

None.

Section 14: Protocols – USB Support

Additions and Changes

None.

Clarifications and Corrections

Section 14.1: EFI_USB_HC_PROTOCOL.SetState() – Description (Page 14-9)

Make the following two changes in the "Description" section:

1. Change the second sentence of the first paragraph in the "Description" section FROM:

   There are three states defined states for the USB host controller.

   TO:

   There are three states defined for the USB host controller.

2. Change the second sentence of the second paragraph in the "Description" section FROM:

   If a device error occurs while attempting the place the USB host controller into the state specified by *State*, then **EFI_DEVICE_ERROR** is returned.

   TO:

   If a device error occurs while attempting to place the USB host controller into the state specified by *State*, then **EFI_DEVICE_ERROR** is returned.

Section 14.1: EFI_USB_HC_PROTOCOL.ControlTransfer() -- Description (Page 14-11)

Make the following two changes in the "Description" section:

1. Change the first sentence of the fourth paragraph FROM:

   EFI_INVALID_PARAMETER is returned is one of the following conditions are satisfied:

   TO:

   EFI_INVALID_PARAMETER is returned if one of the following conditions is satisfied:

2. In the numbered list in the fourth paragraph of the "Description" section, change item #4 FROM:

   MaxiumPacketLenth is not valid. If *IsSlowDevice* is **TRUE**, then *MaximumPacketLength* must be 1, 2, 4 or 8. If *IsSlowDevice* is **FALSE**, then *MaximumPacketLength* must be 1, 2, 4, 8, 16, 32, or 64.

   TO:

   MaximumPacketLength is not valid. If *IsSlowDevice* is **TRUE**, then *MaximumPacketLength* must be 8. If *IsSlowDevice* is **FALSE**, then *MaximumPacketLength* must be 8, 16, 32, or 64.

Section 14.1: EFI_USB_HC_PROTOCOL.BulkTransfer() – Description (Page 14-14)

Make the following two changes in the "Description" section:

1. Change the first sentence of the fifth paragraph of the "Description" section FROM:

   EFI_INVALID_PARAMETER is returned if one of the following conditions are satisfied:

   TO:

   EFI_INVALID_PARAMETER is returned if one of the following conditions is satisfied:

2. In the numbered list in the fifth paragraph of the "Description" section, change item #3 FROM:

   MaxiumPacketLenth is not valid. The legal value of this parameter is: 1, 2, 4, 8, 16, 32, or 64.

   TO:

   MaximumPacketLength is not valid. The legal value of this parameter is: 8, 16, 32, or 64.

Section 14.1: EFI_USB_HC_PROTOCOL.AsyncInterruptTransfer() – (Page 14-17)

Change the first sentence of the fifth paragraph in the "Description" section FROM:

> **EFI_INVALID_PARAMETER** is returned is one of the following conditions are satisfied:

TO:

> **EFI_INVALID_PARAMETER** is returned if one of the following conditions is satisfied:

Section 14.1: EFI_USB_HC_PROTOCOL.SyncInterruptTransfer() – Description (Page 14-20)

Make the following two changes in the "Description" section:

1. Change the first sentence of the fourth paragraph in the "Description" section FROM:

   > **EFI_INVALID_PARAMETER** is returned is one of the following conditions are satisfied:

   TO:

   > **EFI_INVALID_PARAMETER** is returned if one of the following conditions is satisfied:

2. In the numbered list in the fourth paragraph of the "Description" section, change item #4 FROM:

   > *MaxiumPacketLenth* is not valid. The legal value of this parameter is: for the full-speed device, it should be 1, 2, 4, 8, 16, 32, or 64; for the slow device, it is limited to 1, 2, 4, or 8.

   TO:

   > *MaximumPacketLength* is not valid. The legal value of this parameter is: for the full-speed device, it should be 8, 16, 32, or 64; for the slow device, it is limited to 8.

Section 14.1: EFI_USB_HC_PROTOCOL.IsochronousTransfer() – (Page 14-22)

Change the first sentence of the third paragraph FROM:

> **EFI_INVALID_PARAMETER** is returned is one of the following conditions are satisfied:

TO:

> **EFI_INVALID_PARAMETER** is returned if one of the following conditions is satisfied:

Section 14.1: EFI_USB_HC_PROTOCOL.AsyncIsochronousTransfer() (Page 14-24)

Change the first sentence of the third paragraph FROM:

> **EFI_INVALID_PARAMETER** is returned is one of the following conditions are satisfied:

TO:

> **EFI_INVALID_PARAMETER** is returned if one of the following conditions is satisfied:

Section 15: Protocols – Network Support

Additions and Changes

None.

Clarifications and Corrections

Section 15.3: EFI_PXE_BASE_CODE Protocol – Related Definitions – StationIp (Page 15-34)

Add the following sentence to the end of the *StationIp* field description:

> This field must be set to a valid IP address by either **Dhcp()** or **SetStationIp()** before the **Discover()**, **Mtftp()**, **UdpRead()**, **UdpWrite()**, or **Arp()** functions are called.

Section 15.3: EFI_PXE_BASE_CODE Protocol – Related Definitions – SubnetMask (Page 15-34)

Add the following sentence to the end of the *SubnetMask* field description:

> This field must be set to a valid subnet mask by either **Dhcp()** or **SetStationIp()** before the **Discover()**, **Mtftp()**, **UdpRead()**, **UdpWrite()**, or **Arp()** functions are called.

Section 15.3: EFI_PXE_BASE_CODE.Mtftp() – Parameters – BufferSize (Page 15-51)

Replace the following *BufferSize* parameter description:

> For read-file and write-file operations, this is the size of the buffer specified by *BufferPtr*. For read-file operations, if **BufferSize* is smaller than the size of the file being read, then this field will return the required size. For get-file-size operations, this field returns the size of the requested file.

WITH:

> For get-file-size operations, **BufferSize* returns the size of the requested file. For read-file and write-file operations, this parameter is set to the size of the buffer specified by the *BufferPtr* parameter. For read-file operations, if **EFI_BUFFER_TOO_SMALL** is returned, **BufferSize* returns the size of the requested file.

Section 15.3: EFI_PXE_BASE_CODE.Mtftp() – Description (Page 15-53)

Add the following at the end of the second paragraph in the "Description" section:

> Applications using the **`PxeBc.Mtftp()`** services should use the get-file-size operations to determine the size of the downloaded file prior to using the read-file operations—especially when downloading large (greater than 64 MB) files—instead of making two calls to the read-file operation. Following this recommendation will save time if the file is larger than expected and the TFTP server does not support TFTP option extensions. Without TFTP option extension support, the client has to download the entire file, counting and discarding the received packets, to determine the file size.

Section 15.3: EFI_PXE_BASE_CODE.SetParameters() – Parameters – NewSendGUID (Page 15-62)

Add the following sentence to the end of the *`NewSendGUID`* parameter description:

> If *`NewSendGUID`* is **`TRUE`** and there is no SystemGUID, then **`EFI_INVALID_PARAMETER`** is returned.

Section 16: Protocols – Debugger Support

Additions and Changes

None.

Clarifications and Corrections

Section 16.2.1: EFI_DEBUG_SUPPORT_PROTOCOL – Protocol Interface Structure (Page 16-3)

Change the first line of the "Protocol Interface Structure" section FROM:

```
typedef struct _EFI_DEBUG_SUPPORT_PROTOCOL {
```

TO:

```
typedef struct {
```

Section 16.2.1: EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback() – Related Definitions – EFI_SYSTEM_CONTEXT_EBC (Page 16-6)

Change the **EFI_SYSTEM_CONTEXT_EBC** structure FROM:

```
typedef struct {
  UINT64              R0, R1, R2, R3, R4, R5, R6, R7;
  UINT64              Flags;
  UINT64              ControlFlags;
  UINTN               Ip;
} EFI_SYSTEM_CONTEXT_EBC;
```

TO:

```
typedef struct {
  UINT64              R0, R1, R2, R3, R4, R5, R6, R7;
  UINT64              Flags;
  UINT64              ControlFlags;
  UINT64              Ip;
} EFI_SYSTEM_CONTEXT_EBC;
```

Section 16.2.1: EFI_DEBUG_SUPPORT_PROTOCOL.RegisterPeriodicCallback() – Related Definitions – EFI_SYSTEM_CONTEXT_IA32 (Page 16-7)

Change the **EFI_SYSTEM_CONTEXT_IA32** structure FROM:

```
typedef struct {
  UINT32              ExceptionData;  // ExceptionData is
                                      // additional data pushed
                                      // on the stack by some
                                      // types of IA32 exceptions
  EFI_FXSAVE_STATE    FxSaveState;
  UINT32              Dr0, Dr1, Dr2, Dr3, Dr6, Dr7;
  UINT32              Cr0, Cr1, Cr2, Cr3, Cr4;
  UINT32              Eflags;
  UINT32              Ldtr, Tr;
  UINT64              Gdtr, Idtr;
  UINT32              Eip;
  UINT32              Gs, Fs, Es, Ds, Cs, Ss;
  UINT32              Edi, Esi, Ebp, Esp, Ebx, Edx, Ecx, Eax;
} EFI_SYSTEM_CONTEXT_IA32;
```

TO:

```
typedef struct {
  UINT32              ExceptionData;  // ExceptionData is
                                      // additional data pushed
                                      // on the stack by some
                                      // types of IA32 exceptions
  EFI_FXSAVE_STATE    FxSaveState;
  UINT32              Dr0, Dr1, Dr2, Dr3, Dr6, Dr7;
  UINT32              Cr0, Cr1 /* Reserved */, Cr2, Cr3, Cr4;
  UINT32              Eflags;
  UINT32              Ldtr, Tr;
  UINT32              Gdtr[2], Idtr[2];
  UINT32              Eip;
  UINT32              Gs, Fs, Es, Ds, Cs, Ss;
  UINT32              Edi, Esi, Ebp, Esp, Ebx, Edx, Ecx, Eax;
} EFI_SYSTEM_CONTEXT_IA32;
```

Section 16.3.1: EFI_DEBUGPORT_PROTOCOL – Protocol Interface Structure (Page 16-15)

Change the first line of the "Protocol Interface Structure" section FROM:

```
typedef struct _EFI_DEBUG_PORT_PROTOCOL {
```

TO:

```
typedef struct {
```

Section 16.3.3: EFI DebugPort Variable (Page 16-21)

Change the **#define** statements between the second and third paragraphs FROM:

```
#define EFI_DEBUGPORT_ENV_VAR_GUID EFI_DEBUGPORT_PROTOCOL_GUID
```

TO:

```
#define EFI_DEBUGPORT_VARIABLE_NAME L"DEBUGPORT"
#define EFI_DEBUGPORT_VARIABLE_GUID EFI_DEBUGPORT_PROTOCOL_GUID
```

Section 16.4.3: EFI Image Info (Page 16-24)

Change the GUID after the second paragraph FROM:

```
#define EFI_DEBUG_IMAGE_INFO_TABLE_GUID    \
{ 0x49152e77,0x1ada,0x4764,0xb7,0xa2,0x7a,0xfe,0xfe,0xd9,0x5e,0x8b }
```

TO:

```
#define EFI_DEBUG_IMAGE_INFO_TABLE_GUID    \
{ 0x49152E77,0x1ADA,0x4764,0xB7,0xA2,0x7A,0xFE,0xFE,0xD9,0x5E,0x8B }
```

Section 16.4.3: EFI Image Info (Page 16-25)

Make the following changes on page 16-25:

1. Replace the three instances of **UPDATE_STATUS_UPDATE_IN_PROGRESS** with **EFI_DEBUG_IMAGE_INFO_UPDATE_IN_PROGRESS**.

2. Replace the two instances of **UPDATE_STATUS_TABLE_MODIFIED** with **EFI_DEBUG_IMAGE_INFO_TABLE_MODIFIED**.

3. Change the first line of the **EFI_DEBUG_IMAGE_INFO_TABLE_HEADER** structure FROM:

```
typedef struct _EFI_DEBUG_IMAGE_INFO_TABLE_HEADER {
```

TO:

```
typedef struct {
```

4. Change the **EFI_DEBUG_IMAGE_INFO_NORMAL** structure FROM:

```
typedef struct _EFI_DEBUG_IMAGE_INFO_NORMAL {
  UINT32                ImageInfoType;
  EFI_LOADED_IMAGE      *LoadedImageProtocolInstance;
  EFI_HANDLE            ImageHandle;
} EFI_DEBUG_IMAGE_INFO_NORMAL;
```

TO:

```
typedef struct {
  UINT32                      ImageInfoType;
  EFI_LOADED_IMAGE_PROTOCOL   *LoadedImageProtocolInstance;
  EFI_HANDLE                  ImageHandle;
} EFI_DEBUG_IMAGE_INFO_NORMAL;
```

Appendix E: 32/64-Bit UNDI Specification

Additions and Changes

None.

Clarifications and Corrections

Section E.4.7: Initialize (Page E-61)

Add the following sentence to the end of the third paragraph:

> Control will not be returned to the caller and the **COMMAND_COMPLETE** status flag will not be set until the NIC is ready to transmit.

Section E.4.8: Reset (Page E-65)

Add the following sentence to the end of the second paragraph:

> Control will not be returned to the caller and the **COMMAND_COMPLETE** status flag will not be set until the NIC is ready to transmit.

Section E.4.11: Receive Filters (Page E-70)

Add the following sentence to the end of the first paragraph:

> Control will not be returned to the caller and the **COMMAND_COMPLETE** status flag will not be set until the NIC is ready to receive.

Appendix H: Compression Source Code

Additions and Changes

Appendix H: Compression Source Code – AllocateMemory() (Page H-8, Line 3)

Change line three on page H-8 FROM:

```
mText = malloc (WNDSIZ * 2 + MAXMATCH);
```

TO:

```
UINT32      i;

mText = malloc (WNDSIZ * 2 + MAXMATCH);
for (i = 0; i < WNDSIZ * 2 + MAXMATCH; i ++) {
  mText[i] = 0;
}
```


Clarifications and Corrections

None.

Glossary

Additions and Changes

None.

Clarifications and Corrections

Glossary (Page Glossary-11)

Remove the entry for “PCI Host Bridge I/O Protocol.”